AC-209 Final Project
Written Report

Group 49

Andrew Sullivan, James Liounis, and Kane Norman

Table Of Contents

Note: *After exploring the BETH dataset (our assigned topic), we discovered inconsistencies in the data collection methods which reduced the variability of the majority of the features (as the test dataset was the only dataset that contained a confirmed cyber attack, and nearly all the features were related to that single attack). After discussing our worries about the applicability of our future research using that dataset with Chris Gumb, we took his advice to find a new dataset relating to cybersecurity to continue our research. Thus, the following analysis pertains to the* new dataset.

1. Motivation, Context, and Framing

Cyber threats are becoming increasingly widespread, affecting individuals, businesses, and governments around the world. Fraudulent URLs play a central role in cybercrime, as they are often used to trick individuals into visiting fake websites and divulging sensitive information, making them an increasing threat to individuals and organizations alike. Defacement refers to the unauthorized alteration of the appearance of a website, often for the purposes of vandalism. This can damage the reputation of the affected website and the organization it represents. Phishing is a technique that uses fraudulent emails or websites to trick individuals into revealing sensitive information, such as passwords or financial data. This can lead to identity theft or other forms of fraud. Malware, or malicious software, is a broad term that refers to any software designed to harm or exploit a computer system. Malware can be delivered through various means, such as email attachments, downloads from the internet, or by visiting a malicious website. Malware can allow attackers to gain unauthorized access to a computer, steal sensitive information, or even take control of the entire system. In summary, these types of attacks can be extremely harmful to both individuals and organizations and it is important to be aware of these threats and take steps to protect oneself and one's organization against them.

However, simply being aware of and even thoroughly educated on several cyber security threat mitigation measures does not completely mitigate one's exposure to cyber threats. In fact, according to a recent survey by Intermedia, when it comes to phishing attacks, a quarter of IT security staff workers admitted to falling for a phishing scam, compared to one in five office workers (21%).[1] Thus, it has become clear to our group that the capability for humans to detect cyber threats - especially intricacies in the construction of a URL that may indicate a malicious website, is not sufficient; furthermore, predictive models seem poised to play a central role in the detection of cybercrime for years to come.

The purpose of our project is to construct a simple tool to detect fraudulent URLs. Creating a predictive model that tells users if a URL is safe is useful because it can help protect users from falling victim to these types of attacks. This would enable users to input a link that they would come across on the internet or one that they had been sent, in order to assess the risks encountered by clicking on it. By using machine learning algorithms, the model can analyze the characteristics of a given URL and determine the likelihood that it is malicious by identifying its predicted type. This would help mitigate the risks carried by surfing the internet by helping users avoid clicking on potentially harmful links and keep their information safe.

---

[1] https://www.hipaajournal.com/study-phishing-awareness-data-security-training/

2. Dataset Overview

The Malicious URLs Dataset consists of 651,191 URLs, which were classified as benign (or safe), defacement, phishing, or malware - with counts of 428,103, 96,457, 94,111, and 32,520 respectively. The dataset was intentionally gathered to over-represent examples of malicious URLs to aid in the creation of machine learning-based models aimed to identify malicious activity. The data was collected from various sources; the ISCX-URL-2016 dataset[2] was used to collect benign, phishing, malware, and defacement URLs, the Malware domain blacklist dataset[3] was used to gather additional phishing and malware URLs, the Faizan git repo[4] was used to gather additional benign URLs, and lastly, the Phishtank[5] and PhishStorm[6] dataset's were used to further increase the number of phishing URLs.

The response variables used in our analysis are binary indicators labeled "benign", "defacement", "malware", and "phishing." The label "benign" represents a URL that has no malicious content, "defacement" refers to the visual alteration of websites by attackers, "malware" represents a URL that was created with the intent to distribute malicious content such as ransomware, and "phishing" indicates a URL that is socially engineered to pose as a familiar site to gather sensitive information once clicked; we feature-engineered a value of 1 for each of these variables represents the presence of such malicious content for a given URL - with 0 denoting its absence. Each row in the dataset corresponds to a specific URL, and every URL was intentionally gathered to represent a single type of website: "benign", "defacement", "malware", or "phishing" - with no URLs representing a combination of these types.

The initial dataset only included two columns - "URL" and "type" (with "type" being a classifier of the type of content contained in the website: benign, defacement, phishing, or malware). Thus, we spent a considerable amount of time preprocessing the data in the form of additional engineering features based on the original sole feature - URL; these features are described in Table 1 below.

---

[2] https://www.unb.ca/cic/datasets/url-2016.html
[3] http://www.malwaredomains.com/wordpress/?page_id=66%20-%20Deprecated
[4] https://github.com/faizann24/Using-machine-learning-to-detect-malicious-URLs/tree/master/data
[5] https://www.phishtank.com/developer_info.php
[6] https://research.aalto.fi/en/datasets/phishstorm-phishing-legitimate-url-dataset

Table 1: The description and type of each engineered feature

Feature	Type	Description
url_length	int	A count of the total number of characters in a url
@	boolean	Returns 1 if "@" exists in the URL, and 0 otherwise
?	boolean	Returns 1 if "?" exists in the URL, and 0 otherwise
-	boolean	Returns 1 if "-" exists in the URL, and 0 otherwise
=	boolean	Returns 1 if "=" exists in the URL, and 0 otherwise
.	boolean	Returns 1 if "." exists in the URL, and 0 otherwise
#	boolean	Returns 1 if "#" exists in the URL, and 0 otherwise
%	boolean	Returns 1 if "%" exists in the URL, and 0 otherwise
+	boolean	Returns 1 if "+" exists in the URL, and 0 otherwise
$	boolean	Returns 1 if "$" exists in the URL, and 0 otherwise
!	boolean	Returns 1 if "!" exists in the URL, and 0 otherwise
*	boolean	Returns 1 if "*" exists in the URL, and 0 otherwise
,	boolean	Returns 1 if "," exists in the URL, and 0 otherwise
//	boolean	Returns 1 if "//" exists in the URL, and 0 otherwise
abnormal_url	boolean	Returns 1 if a host name (a string containing the domain name) exists in the URL, and 0 otherwise
https	boolean	Returns 1 if the URL indicates a secure website, and 0 otherwise
phishing_words_count	int	The count of how many parsed individual words contained in the URL were among the top 10 most commonly used words in URLs of type "phishing" - see figure 5.
defacement_words_count	int	The count of how many parsed individual words contained in the URL were among the top 10 most commonly used words in URLs of type "defacement" - see figure 5.
malware_words_count	int	The count of how many parsed individual words contained in the URL were among the top 10 most commonly used words in URLs of type "malware" - see figure 5.
benign_words_count	int	The count of how many parsed individual words contained in the URL were among the top 10 most commonly used words in URLs of type "benign" - see figure 5.
tld	str	Top-level domain

Lastly, after our exploratory data analysis (the results of which are described in the visualizations section below and within our Jupyter Notebook), we split the 651,191 data entries in an 80/20 *stratified* train-test split to prepare for modeling.

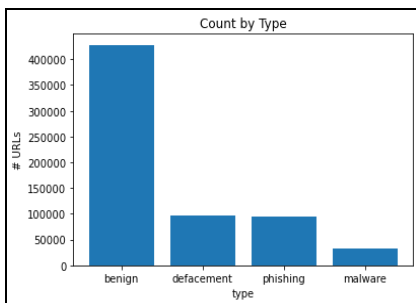# **3.** Exploratory Data Analysis (EDA)

EDA is an essential component of any data science project. It involves analyzing and summarizing data to identify patterns, trends, and relationships, as well as potential issues, anomalies, or missing data. In our project, EDA helped us identify classification trends within the data, which were useful for engineering relevant features and developing more effective and accurate models. Below are some of our initial findings when exploring our dataset.
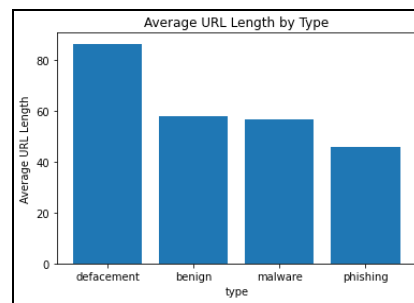
Figure 1: depicts the imbalance of the dataset. The majority of the URLs were classified as "benign", and "malware" URLs were observed much less relative to the other types of malicious URLs.

Figures 2 and 3: Depicts the distribution of URL length based on the type of URL. Figure 2 displays how "defacement" URLs were longer on average than the other types, and the violin plot shown in figure 3 highlights the existence of outliers in the form of extremely long lengths for URLs of type "phishing" and "benign."
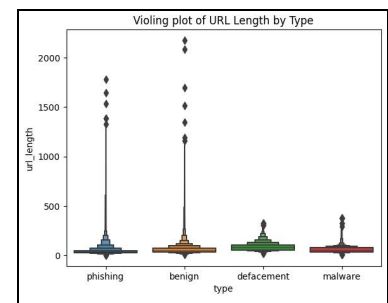
| Figure 1: Bar, Count of URLs by Type	Figure 2: Bar, Average URL Length by Type	Figure 3: Violin, Average URL Length by Type

Figure 4: depicts the top-level domains (TLD) that exist within the top 85% of URLs for each given type. As one may expect, '.com' is the most frequently occurring TLD across all types. '.org' appears to be a more important predictor for "phishing" and "benign" URLs relative to other types. Interestingly, TLDs that represent a URL's country of origin (such as '.gov', '.uk', '.ca', etc.) appear more frequently in 'defacement' and 'phishing' types. Another valuable insight observed from this plot is gained from the TLD 'info' which does not appear within the top 10 most frequently occurring TLDs for any type other than for malware - where it ranks 2nd in frequency behind '.com'.

Figure 5: shows the most frequently appearing words based on the type of URL. From inspection, we can see that certain words appear more frequently in one category than others. For example, the phrase "pastehtml" commonly appears in phishing URLs but not others. A simple google search will show that "pastehtml" is increasingly abused by phishers. This suggests that the presence of certain words in a URL may help indicate which group it can be classified into.

Figure 4:
Most frequent TLDs by
Type of URL
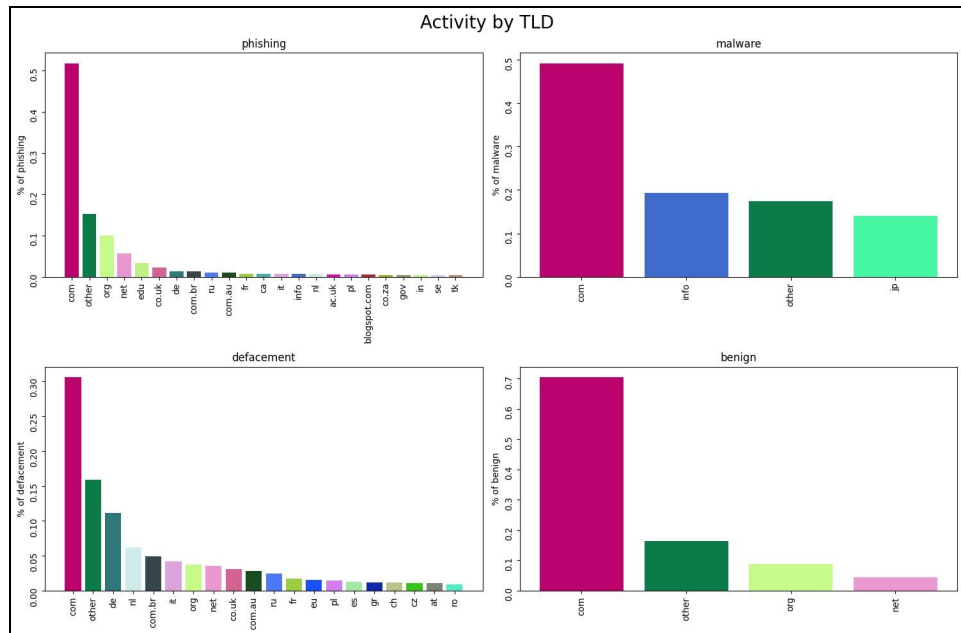
Figure 5:
Top 10 Most Frequent
URL Words by Type
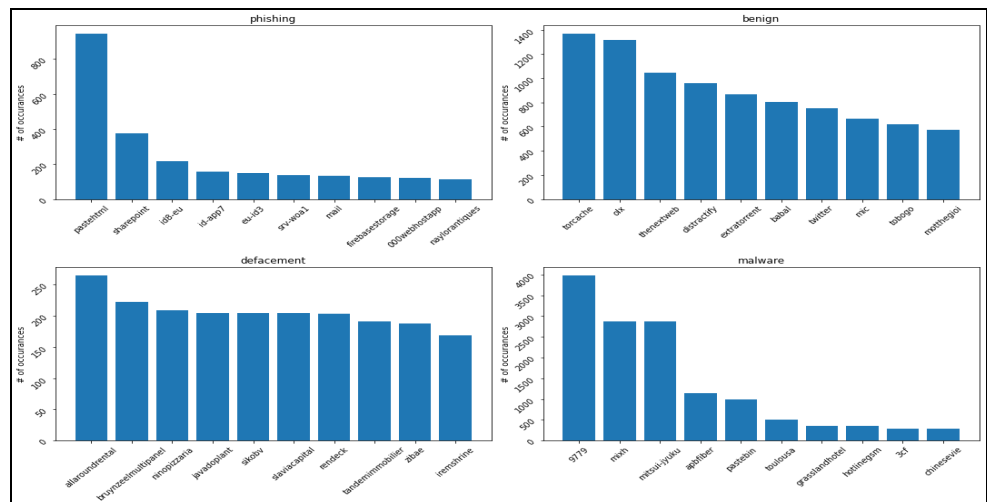
4. Revised Project Question

The aim of our project is to develop a classification model that helps predict whether a URL contains benign, defacement, phishing, or malware content based on a range of features. We are conducting a two-part analysis: we start with EDA, feature engineering, and the creation of naive baseline models. Subsequently, we will later attempt to improve our baseline model performance by hyperparameter tuning and other methods.

5. Train-Test Split

In this project, we train a classification model on an imbalanced dataset. Oversampling is a popular method for dealing with imbalanced data, but it can also introduce bias and reduce generalizability. This is because oversampling creates synthetic examples of the minority using existing examples. These synthetic examples may not be representative of the true distribution of the original data, which can cause the model to generalize poorly. Additionally, oversampling can make it difficult to accurately evaluate the performance of a model. This is because the test set is not representative of the original data distribution, which can cause the model to appear more accurate than it actually is.

Ethically, we are obliged to ensure that we are fairly interpreting the accuracy of our model. We don't want to mistakenly present a weak model to our end users who may be relying on our results to protect the safety, security, and privacy of themselves and their loved ones.

Overall, while oversampling can improve the performance of a model on the training set, it can also introduce bias and make it difficult to evaluate the model's accuracy. For these reasons, we opted for a stratified train-test split to handle the imbalance in our dataset.

6. Baseline Model Selection

We are dealing with a classification problem, so we narrowed our potential baseline models to logistic regression, decision tree classifier, and random forest classifier. We decided not to attempt training a KNN classification model because the computational complexity of the KNN model would take too long to run for our short time constraint. Also, KNN is prone to overfitting when dealing with high dimensionality.

We started with a one-vs-many approach where we categorized each URL into one of the four possible categories by training a separate logistic regression model for each category. Each model was trained using the same stratified training data and features. The results obtained relatively high accuracy scores, but the class imbalance in our dataset raised some legitimate concerns. For example, the heavy imbalance of "benign" URLs gives our model the opportunity to categorize each URL as 'benign" and still score a high accuracy. From an ethical viewpoint, publishing a model that falsely over-classifies URLs as "benign" would jeopardize the safety and security of end users.

Additionally, we were concerned that using a large generic set of predictors for each model in the one-vs-many approach for a multi-classification problem could potentially increase the complexity of our model, leading to overfitting noise in the high-dimensional training set. This is because certain features may not have consistent predictive power across each model, causing us to include additional weak and complex features.

We confirmed our suspicions by applying a regularization penalty to our one-vs-many classification models. Specifically, after applying the regularization penalty, the accuracy of our model for "phishing" was diminished by over 50%. This drop in performance suggests that we would need to train each model on a smaller set of relevant features, but this would require us to perform additional feature engineering tailored to each category.

Given this dilemma, we have two options: we can either incur the overhead of additional feature engineering and follow through with the one-vs-many approach, or we can train multiclass models on our dataset. Each approach has its advantages and disadvantages, but after careful consideration, we decided it was best to explore models better suited for multiclass response variables.

We started with an untuned multinomial regression model which performed worse than the following two models. We then proceeded to implement the Scikit-Learn Decision Tree Classifier. This resulted in a test accuracy of 0.88. Our confusion matrix below depicts a strong classification performance, with the exception of the predictions for the phishing and benign categories (figure

6 shows that 2,827 benign URLs were wrongly classified as phishing, and 8,578 phishing URLs were classified as benign). Despite this, our classification report boasts an average precision score of 0.88.

We then implemented the Random Forest Classifier. Recall that a random forest can often provide better performance than a single decision tree. This is because a random forest is able to capture complex decision boundaries in the data, while a single decision tree cannot. Additionally, a random forest is less likely to overfit the data than a single decision tree, which makes it a more reliable model for making predictions on new data. Fitting an untuned Random Forest Classifier gives us an accuracy of 0.889, improving marginally from the decision tree. Similar to the decision tree, the model experiences confusion between phishing and benign. Our precision score remains unchanged, averaging approximately 0.88.

After our initial modeling process, we are pleased with the performances of both these models. We select the Decision Tree Classifier as our baseline model.

Figure 6:
Confusion Matrix for Decision Tree Classifier

Figure 7:
Confusion Matrix for Random Forest Classifier
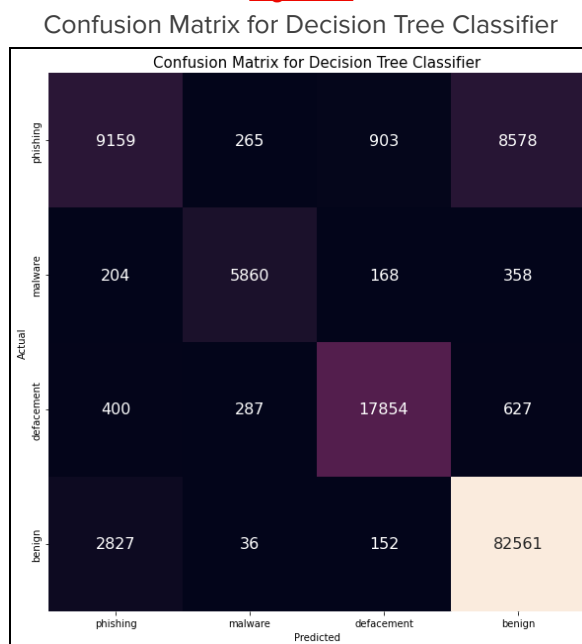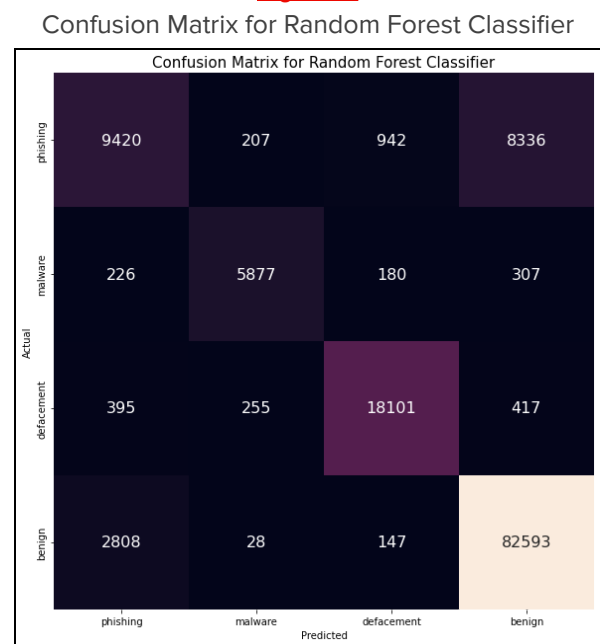
7. Modeling Approach

The main goal of our modeling approach was to accurately estimate complex decision boundaries without overfitting. Since our multi-classifier and ensemble methods showed promising performance in the previous step, we decided to fine-tune these models to further improve their accuracy and F1 score. Note that we are paying attention to both metrics: they are both used to evaluate the performance of a classification model. The main difference between these two metrics is that accuracy is a ratio of the number of correct predictions made by the model to the total number of predictions, while the F1-score is a weighted average of the precision and recall of the model.

Our first endeavor was to tune the stopping conditions of our baseline model, starting first with its maximum depth. To this end, we performed 5-fold cross-validation and obtained the highest mean cross-validation accuracy for a max-depth of 23 (fitting trees of depth 1 to 24 suggest that this metric decreases after this threshold). However, this only led to a minor increase in accuracy (88.59, 0.3% increase from baseline). Also, this created an additional concern: not specifying `max_leaf_nodes` led to our decision tree performing a "depth-first" search, which is nonoptimal. We perform a grid-search in order to find the optimal parameter, but also take the opportunity to tune other stopping conditions such as min_sample_leafs and min_impurity_decrease, which can only help the classification process. The searching process then switches from "depth-first" to "best-first".

This suggested that we needed to pursue further exploration of the hyperparameters, which we achieved via a grid search.

After a 5-hour run, this grid search did not deliver the desired results, providing us with lower accuracy and F1-score. We hence assumed that this was because the range of provided parameters was not exhaustive enough. We could have provided a more exhaustive list of parameters that would have taken a long time to complete. In the interest of time, we were forced to select the un-tuned model as the preferred one between the two.

We then turned to tune our Random Forest Classifier, which was our best-performing untuned model. Running a similar grid search provided us with a set of hyperparameters that improved its performance. With it boasting an F1-score of 0.886, we assumed it would be hard to surpass it in terms of performance. This led us to serialize it in order to use it later on for our GUI (please refer to `detection-interface.py`).

For the sake of exploring every possible avenue, we invoked desirable properties of XGBoost and AdaBoost and considered them for our analysis.

XGBoost may be preferable over Random Forest because it uses gradient boosting to improve the predictive power of the model. This means that XGBoost is able to take into account the previous predictions made by the model when making new predictions, which can increase its accuracy.

As for the latter, AdaBoost can sometimes provide better performance because it is able to focus on the observations that are most difficult to predict, which can improve the overall predictive power of the model.

We first trained an un-tuned version of both models, in turn tuning them both with gridsearches. Tuning for the learning rate in XGBoost led it to achieve an F1-score of 0.885. Tuning for the learning rate, the number of estimators, and the base estimator for AdaBoost (Decision Tree with varying depths, in this case) increased its F1-score by roughly 3% compared to its baseline accuracy. Its highest F1 score was 0.794.

8. Going Further (Part 1): Implementing the Multi-Layer Perceptron Model

As we conducted background research on the topic of URLs and cybersecurity, we came upon some insightful and applicable research conducted by Mohammad et al. (2014). What started as a simple attempt to gather useful insight into their research about predicting phishing websites ended with a fascination with the unique self-structuring neural network architecture that they deployed. So much so, that it motivated us to implement one ourselves.

Mohammad et al. (2014) utilized a feed-forward neural network architecture with one hidden layer, which is commonly known as multilayer perceptron. As explained by Mohammad, one of the main advantages of multilayered perceptrons is that "the number of neurons in the hidden layer can be changed to adapt to the complication of the relationships between input and output variables" (Mohammad et. al., 451). In our implementation of this model, we set hidden_layer_sizes=(26,) - denoting only 1 hidden layer with 26 hidden neurons. These specifications were made based on Sarle's (2002) comments on neural networks. According to our research, 1 hidden layer is a commonly accepted starting point when it comes to tuning the MLPs, however, as we discussed, we were limited in our ability to grid-search this parameter. The size of the layer was determined by calculating the mean of the neurons in the input - the number of neurons comprising this layer is equal to the number of features, and output - a single node because our neural network is a classifier, layers. Supported by the understanding that MLP classifiers are known for their ability to learn and make predictions based on patterns in the input

data, we theorize that this model may be effective in its ability to recognize the characteristics of malicious URLs and make predictions about the likelihood that a given URL is malicious; thus, we implemented it as such in our research.

Additionally, we select the L-BFGS solver. On the one hand, an advantage of using the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) solver for training a multi-layer perceptron (MLP) is that it typically converges faster than other first-order optimization algorithms, such as stochastic gradient descent (SGD). On the other hand, the L-BFGS solver is well-suited for high-dimensional optimization problems, which is the case with our dataset. LBGFS is a quasi-Newton method: it uses an approximation of the Hessian matrix to make updates to the model's parameters, rather than computing the Hessian. Quasi-Newton methods take the following general form:

$$x_{k+1} = x_k - \alpha_k \, B_k^{-1} \nabla f(x_k)$$

Results: in this case, the MLP Classifier performs significantly well, but still worse than the Random Forest Classifier in regard to both accuracy (0.819) and F1-score (0.798). This may be because it is unable to capture the same level of complexity as the Random Forest, and is more prone to overfitting, hindering its ability to generalize well to new data (Montantes, 2020). Finally, the type of features which make up our dataset may explain this result, as the MLP Classifier is more appropriate for continuous features.

9. Going Further (Part 2): User Interface

Our model has the ultimate goal of helping internet users avoid cyber threats that can be difficult to identify, even for well-trained individuals. To achieve this, we developed a user-friendly interface that interacts with our pre-trained (and serialized) Random Forest model. This interface allows users to enter a website link and quickly receive a prediction of the safety of the website. The user interface can be easily used as needed. By providing users with a simple tool for evaluating website safety, our model and user interface can help protect users from potential cyber threats and allow them to make informed decisions about the websites they visit.

10. Results

	Test Accuracy	F1-Score
Fine-Tuned Random Forest Classifier	0.891507	0.886268
Fine-Tuned Decision Tree Classifier	0.886079	0.880480
Fine-Tuned XGBoost	0.885848	0.885445
Fine-Tuned AdaBoost	0.828323	0.772100

The Random Forest Classifier was the best-performing model, achieving a test accuracy of 0.891 and an F1-Score of 0.886. This superior performance can be attributed to several factors. Firstly, the use of an ensemble of decision trees allows the model to capture complex decision boundaries and make more reliable predictions. This is particularly relevant to address the subtle difference between malicious URL categories. Additionally, the ability to handle high-dimensional data with subtle differences between classes makes it well-suited to this particular dataset. Combining random subsets of the data also helps to reduce overfitting and improve the generalizability of the model. These factors likely contributed to its superior performance compared to the Decision Tree, even after tuning its stopping conditions. In the future, higher computing power may enable us to conduct a more thorough hyperparameter tuning process and potentially improve the performance of the Decision Tree.
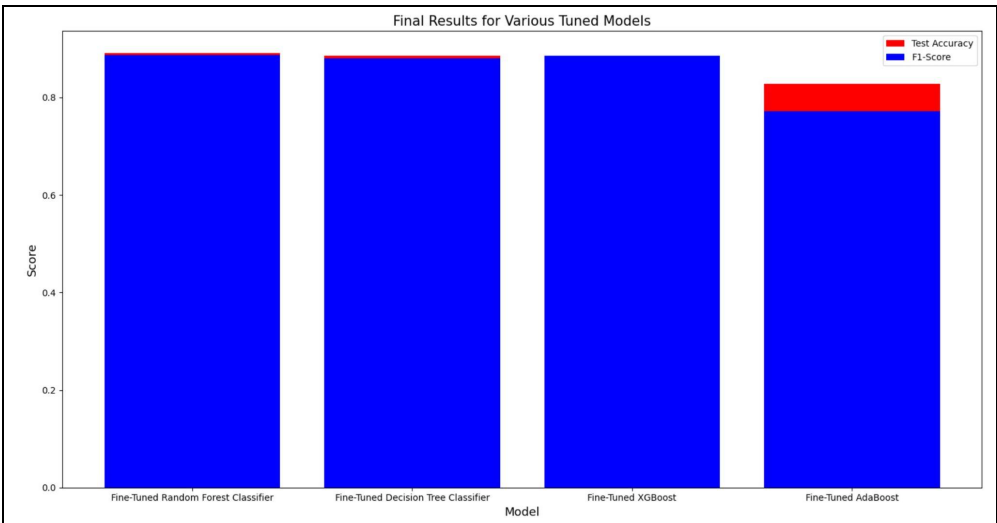
The Random Forest Classifier is particularly well-suited to avoid overfitting in this case. This is because each decision tree in the ensemble is trained on a different subset of the data, using a random selection of features for each split. This means that the individual trees are less likely to overfit the training data, and the final prediction is made by averaging the predictions of all the trees. This ensemble approach makes the random forest classifier less sensitive to noise and outliers in the data, which can improve its overall performance. This may explain why it outperformed XGBoost and AdaBoost in this scenario.

Also, one of the key advantages of the Random Forest Classifier is its ability to handle both continuous and categorical features. This is particularly important in this case, where the data may have a mix of different feature types as a result of the extensive feature engineering process. Additionally, the use of an ensemble of decision trees allows the model to be less sensitive to noise in the data. This makes it well-suited to the current dataset and likely contributed to its superior performance compared to XGBoost and AdaBoost.

11. Limitations and Future Work

A significant issue that we came across while performing hyperparameter tuning was that some of our tuned models performed worse than our untuned/partially-tuned ones. We assumed that this was simply because our classifiers were already performing strongly, and the list of hyper-parameters they were told to select from was not exhaustive enough.

We could have given our grid searches more hyperparameters to choose from. However, our grid searches were already way too computationally expensive as such, with an average of 5+ hours to run each grid search on a powerful 10-core CPU, 32-core GPU, 16 core neural engine machine.

What we would like to do in a future iteration of our project is to run these algorithms in a High-Performance-Computing setting, using for example Amazon Sagemaker from the AWS Machine Learning family. Sagemaker enables you to mount a Jupyter Notebook on a server with a CPU of your choice. Running hyperparameter tuning in the Cloud would then enable us to consider a wider range of hyperparameters, thus significantly increasing our chances to improve the F1 scores and accuracies of our already-high-performing models.

12. References

Ma, J., Saul, L.K., Savage, S. and Voelker, G.M. (2011). Learning to detect malicious URLs. ACM
Transactions on Intelligent Systems and Technology, 2(3), pp.1–24.
doi:10.1145/1961189.1961202.

Ghaleb, F.A., Alsaedi, M., Saeed, F., Ahmad, J. and Alasli, M. (2022). Cyber Threat
Intelligence-Based Malicious URL Detection Model Using Ensemble Learning. Sensors,
22(9), p.3373. doi:10.3390/s22093373.

Mohammad, R.M., Thabtah, F. & McCluskey, L .(2014) Predicting phishing websites based on
self-structuring neural network. Neural Comput & Applic 25, 443–458.
https://doi.org/10.1007/s00521-013-1490-z

Sarle, W. (2002). ai-faq/neural-nets. [online] www.faqs.org. Available at:
http://www.faqs.org/faqs/ai-faq/neural-nets/part1/index.html.

Montantes, J. (2020), 3 Reasons to Use Random Forest Over a Neural Network–Comparing
Machine Learning versus Deep Learning.
https://towardsdatascience.com/3-reasons-to-use-random-forest-over-a-neural-network-
omparing-machine-learning-versus-deep-f9d65a154d89